THE COMMERCIALIZATION OF A RAPID PROTOTYPING TOOL FOR SIMULATING COMMAND AND CONTROL APPLICATIONS

Report No. IST95-R-085

29 June 1995

Prepared for:

US Army Research Office
Attn: AMXRO-RT-IP (Ramseur)
PO Box 12211
Research Triangle Park, NC
27709-2211

Contract No.

DAAH04-94-C-0053

Prepared by:

Illgen Simulation Technologies, Inc.
250 Storke Road, Suite #10
Goleta, CA 93117

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 29, 1995	Final 30Sep94 to 29Jun95

4. TITLE AND SUBTITLE
The Commercialization of a Rapid Prototyping Tool for Simulating Command and Control Applications

5. FUNDING NUMBERS
DAAH04-94-C-0053

6. AUTHORS
Dr. Steven Chavin
Dr. Valdis Berzins
Dr. Larry Dworkin
Dr. Luqi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Illgen Simulation Technologies, Inc.
250 Storke Road, Suite #10
Goleta, CA 93117

8. PERFORMING ORGANIZATION REPORT NUMBER
IST95-R-085

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
Attn: AMXRO-RT-IP
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 33379.1-RT-ST1

11. SUPPLEMENTARY NOTES
This work sponsored by the U.S. Army Research Office, Code AMXRO-RT-IP, under DAAH04-94-C-0053

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; STTR report, distribution unlimited.

12B. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words) This report describes the work done to commercialize the Computer Aided Prototyping System (CAPS). The work that was completed was:

(1) Develop the requirements and the documentation to restructure and reengineer the CAPS code to improve efficiency and utility for commercial development.
(2) Establish Alpha test sites at Illgen Simulation Technologies, Inc. (ISTI), Monmouth University (MU), the MITRE Corporation and the U.S. Army Communications and Electronics Command at Fort Monmouth (CECOM).
(3) Develop a tutorial manual and an installation manual for use at these sites.
(4) Apply CAPS to new applications that will demonstrate the tool's usefulness and the ability of new users to understand and work with CAPS.
(5) Create a commercialization plan that consists of four related sections. These include a technical plan, managerial plan, marketing plan, and a product support plan.

14. SUBJECT TERMS
Computer-Aided Prototyping System (CAPS)
Iterative/Adaptive Graphically Driven Interfaces
Evolution of Software-Intensive Systems

15. NUMBER OF PAGES 42

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	None

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std 239-8
298-102

TABLE OF CONTENTS

TABLE OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1

EXECUTIVE SUMMARY

1.1 Background

The objective of this STTR Phase I program is to begin transitioning an automated rapid-prototyping software development tool into a commercial product. This tool, known as the Computer-Aided Prototyping System (CAPS), is an extensive, feature-rich software development tool that can be used in all phases of the software creation process to support the prototyping, development, and evolution of real-time software systems.

CAPS uses the Prototype Specification Design Language (PSDL), which is a custom programming language that has both a graphical and text-oriented structure. Using PSDL, a designer can produce a prototype of a system to test the architecture and timing constraints. CAPS creates Ada code to connect the pieces of the prototype and uses a sophisticated program to create a schedule for real-time software. CAPS contains execution support modules that allow man-in-the-loop simulation of the software system. Configuration management tools are also part of CAPS.

CAPS has been under development (by Drs. Berzins and Luqi at the Naval Postgraduate School) for more than eight years. The research effort by these professors has produced a powerful laboratory prototype, with many sophisticated software system features, but with none of the necessary features, such as user-friendliness, user documentation and an automated installation procedure, required to transition CAPS into the commercial software marketplace. This work has been supported by National Science Foundation (NSF), Ada Joint Program Office (AJPO) and Army Research Office (ARO) grants.

In Phase I of the STTR, the Illgen Simulation Technologies, Inc. (ISTI) team was formed to move the CAPS prototype out of the laboratory and into the commercial world. To achieve this objective, the team, composed of ISTI, Monmouth University (MU), and consultants, Drs. Berzins and Luqi, developed a plan for that transition. The goal was to utilize the CAPS prototype software as the core of a commercial product and then to make improvements to the user interface and to prepare documentation. Under the Phase I effort, the ISTI team created a CAPS commercialization plan, performed a software requirements analysis, and established α/β CAPS test sites. Section 2 of this report discusses the work that was performed during this Phase I effort.

A proposal was also written and submitted that outlines work to be accomplished during the CAPS STTR Phase II commercialization effort. During this phase, the development of the CAPS software will be completed by adding features expected from a commercial software product. The commercialization plan will be executed during this phase, the

software requirements will be implemented, and the outputs from the α/β test sites will be integrated into the prototype software.

The Phase II plan treats technical, managerial, marketing, and software support efforts. The technical efforts include engineering the architecture, modifying current software algorithms, enhancing the man-machine interface, improving the flexibility and extensibility of the tool, and providing a library of reusable design objects that enhances the development of C, C++, and Ada-coded software. The result will be a series of "released for sale" versions of the tool, starting with Version 1.6 in 1997. Details of the CAPS Phase II commercialization effort are discussed in Section 3 of this report.

Because the commercialization effort will "unlock" some very powerful features, we expect the CAPS commercial product to be highly successful, especially among the developers of real-time software. Real-time software systems are found in both military (C^3I systems, missile seekers, etc.) and commercial (robotics, communication systems, etc.) arenas, and CAPS can be used to support development of these systems. Demand for the software is anticipated to be great.

The marketing of CAPS is a natural extension of ISTI's primary business objectives, i.e., software development and simulation verification and validation (SV&V). As a leader in SV&V, ISTI has been and will continue to be committed to the use of Computer Aided Software Engineering (CASE) tools to aid in the development of problem-free software. SV&V is most effective when it is undertaken concurrently with software development, and CAPS fits well into this approach. ISTI can capitalize on its years of expertise in the SV&V field to make the transition of CAPS into the commercial marketplace a reality.

1.2 New Material in This Report

Much of the material included in this final report can be found in our STTR Phase II proposal, "The Commercialization of a Rapid Prototyping Developmental Tool for Real-Time Embedded Software Intensive, Process and Resource Management Systems". (The Phase II proposal instructions encourage Phase I participants to use the identical language in the proposal and in the final report, when reporting the work completed in Phase I.) New material, not included in the Phase II proposal, can be found in this report in Sections 2.4.9 and 3.2.

1.3 Organization of This Report

The organization of this report is as follows: In Section 2 there is a description of the Phase I objectives and the approach taken to achieve these objectives. The results of the Phase I work are described in Section 2.4. Section 3 describes the technical reengineering plan and the future CAPS product line -- the two most important results of the Phase I work. Appendix A describes the motivation behind the creation of the CAPS product. Very specific and glaring problems were seen by Drs. Luqi and Berzins when they started

the CAPS project over eight years ago at the NPS. We have highlighted these problems in Appendix A. These two individuals designed CAPS with solutions to these problems in mind and Appendix A is designed to provide background for the reader.

SECTION 2

PROJECT SUMMARY PHASE I

2.1 Background

On 1 October 1994 the Army Research Office awarded a Phase I STTR to a team headed by ISTI with support from MU and consultants, Drs. Luqi and Valdis Berzins. *The Commercialization of a Rapid Prototyping Tool for Simulating Command and Control Applications* had a value of $100K, with a 9-month duration, ending 30 June 1995. This Phase I STTR dealt with a rapid prototyping tool called the Computer Aided Prototyping System (CAPS) that was developed at the NPS in Monterey, CA.

The CAPS approach offered a true prototyping language and architectural framework that combines the best features of a computer aided software tool and man-in-the-loop simulation. It provided a mechanism for requirements refinement via iterative, build-a-little, test-a-little philosophy, concept confirmation from the rapid creation and test involving the real user, and the creation of Ada code for rapid insertion into real equipment platforms. CAPS presently consists of a user interface module, a reusable software database system, and a flexible execution support system. Its power lies in its ability to automatically (1) execute program generation, (2) conduct analysis of design scheduling, (3) generate real time processor scheduling, and (4) provide output code once the scheduling exists. As a result, code that incorporates global communication and real time control are automatically generated. This capability overcomes the uncertainty and brittleness that is intrinsic in manually produced real time software code.

In addition, CAPS supplies automatic assistance for including reusable code. This is accomplished via keyword search, subtype matching, and semantic matching that confirms a match to the software specification and assists on certification of the code. The software database also supplies a design history trace that assists in configuration management.

While CAPS has considerable potential for application to a wide variety of problems, little work has been done to prepare it for export to the commercial market place. The team of ISTI and MU, using the expertise of consultants Drs. Luqi and Valdis Berzins, who led the initial development of CAPS, proposed under the STTR program to rectify this shortfall. This team provided for the improvement, testing, and documentation necessary to make CAPS a successful marketable product. This product also has considerable potential for responding to the many shortfalls in command and control (C^2) models existing with Distributed Interactive Simulation (DIS). The program was proposed to be performed in two phases. The Phase I STTR effort established the framework, to include user feedback and documentation, that identified the necessary requirements for a versatile tool

that has both military and commercial applications. Phase II will take the results of Phase I and conduct the research and development necessary to yield a well-defined deliverable product suitable for commercialization and export to other potential user groups.

The overall CAPS architecture is shown in Figures 1 and 2.

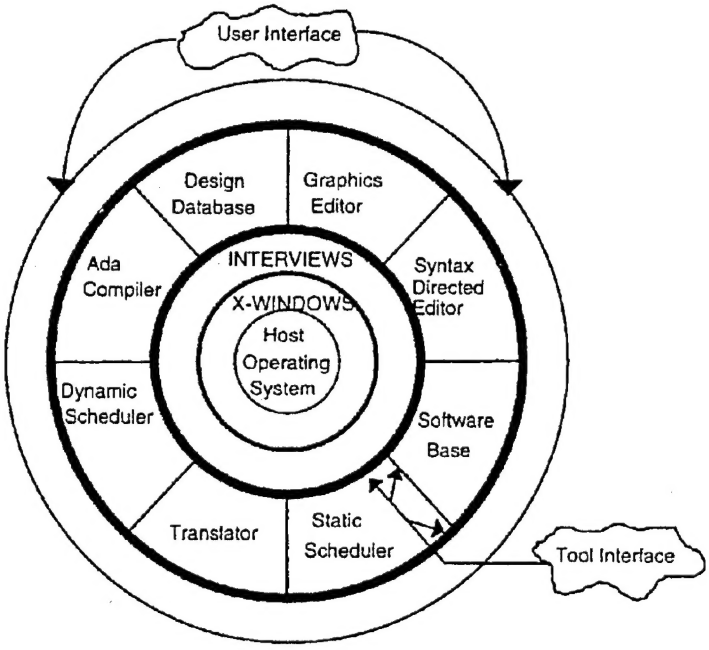


Figure 1. The CAPS Environment.

The tool is capable of supporting all phases of software development for real-time embedded software systems. These include requirements analysis, design, development, implementation conceptual testing, and configuration management.

2.2 Phase I Objectives

The following is a list of objectives for the Phase I program.

- Develop the requirements and documentation necessary to commercialize the tool.
- Export CAPS as a suitable rapid prototyping tool.
- Seek a broad cross section of military and commercial C^2 type applications.
- Analyze current capabilities and limitations.

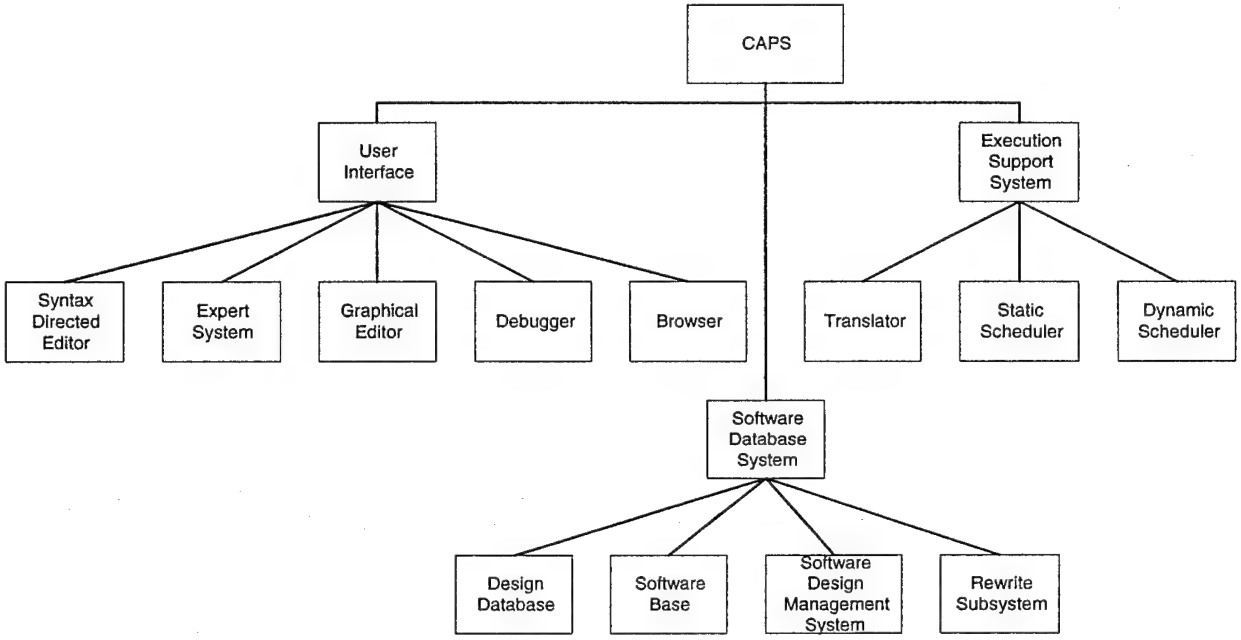


Figure 2. Main Computer-Aided Prototyping System Tools.

- Develop an initial set of requirements for restructuring and reengineering.
- Review architecture and source code as necessary, to improve efficiency and user interface.
- Develop a plan of enhancements that uses α/β test sites for interactive development and continued use in Phase II.
- Develop preliminary user's manuals for α test sites.
- Develop a "proof of concept" demonstration.
- Prepare a plan for commercialization for use during the Phase II portion of the effort.
- Develop a list of prospective customers.

Each of these objectives has lead to specific results, as discussed in Section 2.4.

2.3 Phase I Approach

The effort was conducted in six tasks spanning the 9-month period. These tasks are enumerated below, along with a brief outline of a general support environment.

- Task 1. Install CAPS at ISTI and MU.
- Task 2. Develop a plan for code restructuring to improve efficiency and utility.
- Task 3. Develop and deliver a preliminary user's manual.
- Task 4. Develop an enhancement plan for commercialization of the CAPS tool.
- Task 5. Establish α test sites at MITRE and CECOM.
- Task 6. Supply CAPS to selected C^2 applications.

- Establish a common hardware and software environment.
- Set up distributed collaborative development environments.

2.4 Results of the Phase I Effort

The objectives outlined in Section 2.2 have produced a series of documents, α site hardware and software facilities, and plans that will be used in Phase II of this effort. This section describes each of these in greater detail.

2.4.1 Key Requirements

In prior work at NPS (Reference 1), Drs. Berzins and Luqi outline a set of requirements for CAPS. This list was augmented by ISTI and MU with special emphasis on those requirements associated with an industrial commercial tool. Table 1 summarizes these key requirements.

These requirements are associated with a commercialized rapid prototyping and software development tool. CAPS is unique in that it has a number of architectural elements that addresses key functions needed in software development. In Phase II of the STTR program, these requirements will be satisfied through a series of release versions over the next several years.

2.4.2 Exporting CAPS to New Locations

The establishment of α sites at locations outside the NPS had several objectives. These include:

1. To begin the export of the tool to a wider cross section of users.
2. To assist in the evolutionary evaluation and development of CAPS.
3. To gain hands-on experience with CAPS as a software development tool for the four α site users.
4. To establish a linked distributed CAPS environment that will be carried into Phase II as β sites.

2.4.2.1 Development Environment. MU was established, during Phase I, as the hub of the CAPS development environment, as shown in Figure 3.

A SunSPARC 5 at MU, loaded with the required CAPS elements, is accessible via a dial-up modem and SLIP protocol via the telephone lines. A Telnet access is also available via the Internet. All locations are linked via e-mail and an electronic bulletin board.

Table 1. Requirements for a successful general purpose software development CASE tool for real-time embedded systems.

Requirement	Function
The CASE tool should implement a software development environment that supports the entire life cycle process, from software design and testing through revision and final product release.	***The tool should assist the software designer/analyst in efficiently and effectively developing a software end-product.***
1. The tool environment should support configuration management and change control functions.	Configuration management and change control functions will provide the user with the methods and tools for: ♦ Identifying, storing and assessing software components. ♦ Developing a documentation repository. ♦ Automating support for change requests. ♦ Generating a development history audit trail. ♦ Examining traceability between components and product evolution.
2. The tool environment should support project management functions.	Project management functions will provide the user with the methods and tools for: ♦ Automatically generating project status reports. ♦ Automatically rescheduling project workplans. ♦ Automatically briefing programmers.
3. The tool environment should support a common/shared repository for the maintenance of design documentation and software code.	A common/shared repository of design documentation and software code will provide the user with: ♦ Version control over both software code and documentation. ♦ Global consistency and completeness checking.
4. The tool environment should consider human factors.	Human factors consideration will allow the user to: ♦ Access on-line system and team member interaction. ♦ Eliminate complex commands. ♦ Intuitively assess system operation. ♦ Access on-line diagnostics and on-line help.
5. The tool environment should support real-time application functions.	Real-time application functions will provide the user with: ♦ Methods and tools for generating real-time system schedules.
6. The tool environment should support automatic code generation from a graphical specification language	Automatic code generation from a graphical specification language will allow the user to: ♦ Generate a pictorial view of the software. ♦ Automate software generation tasks to produce Ada, C, C++ code ♦ Offer predesigned applications
7. The tool environment should support the characterization and reuse of software modules with a populated database.	Software module characterization and reuse functions will: ♦ Reduce programming workload. ♦ Increase programmer competence.
8. The tool environment should support rapid integration functions for the integration of developed software with other software packages.	Rapid integration functions will provide the user with a wide spectrum of tools for: ♦ Automating software integration tasks.
9. The tool environment should support rapid software prototyping and design testing functions.	Prototyping and design testing functions will provide the user with the methods and tools for: ♦ Rapid functional prototyping, performance prototyping and behavioral prototyping. ♦ Rapid testing of real-time systems.
10. The tool environment should support flexibility and extensibility.	Flexibility and extensibility provides the user the ability to: ♦ Use and work with different operating systems and workstation environments.

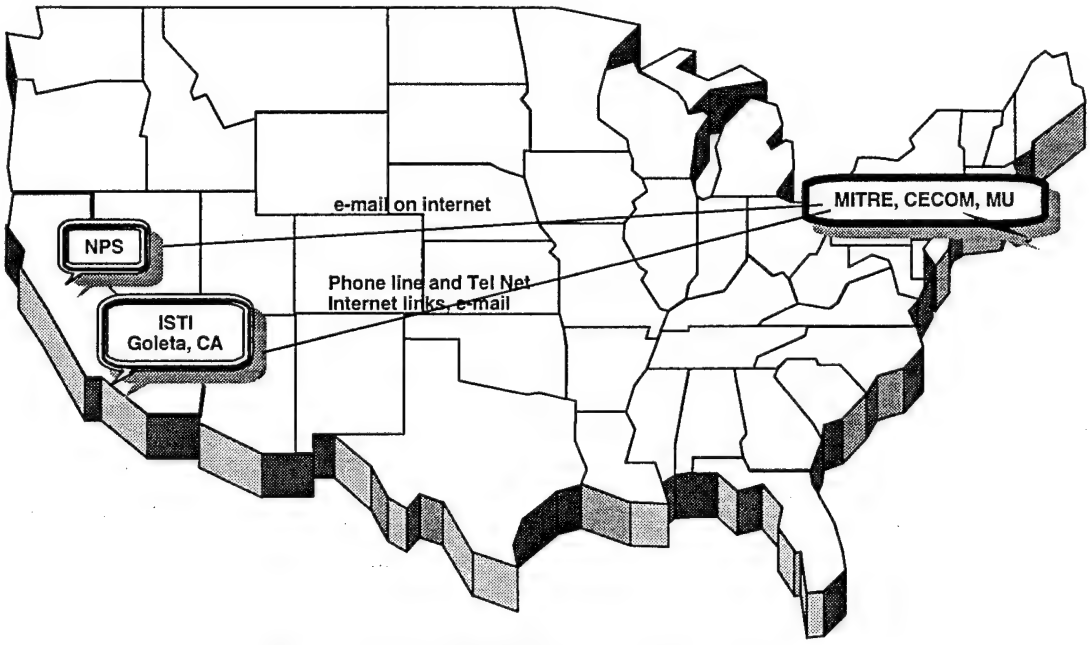


Figure 3. CAPS Development Locations.

The MU site was operational by 3 March 1995 and the MITRE and CECOM links were on-line on 22 May 1995. The ISTI site was operational in January 1995. In addition to the distributed environment, each location can run CAPS within their own hosts. A user help-hot-line was established at MU for assistance to all α sites.

A one-day training course was given on 17 May 1995 by MU faculty and graduate students for MITRE and CECOM personnel. The training consisted of three hours of classroom instruction on CAPS, a one-hour demonstration, and two hours of hands-on use. The access procedure, the building of a simple prototype system using the CAPS Prototype System Description Language (PSDL), and the compiling of the prototype into Ada was completed by all participants. In addition, a user's manual that was developed by NPS and MU was given to all participants.

2.4.3 New Applications of CAPS

Four new applications of CAPS have been examined. These include: (1) an aircraft steering system; (2) a forward-area defense system; (3) a street light traffic control system; and (4) a binary adder. The simplest of the four (applications 1 and 4) were designed, compiled, and run in the CAPS environment. Of special note is that use was made of the existing CAPS object library (part of the temperature controller) already written in Ada. By just renaming the objects and changing the timing constants a graduate student was able to build the next application without any knowledge of Ada. Applications 2 and 3 are currently under design in the CAPS PSDL environment. Application 2 is using the component objects in the C^3I example developed by Drs. Luqi and Berzins

(Reference 1). Application 3 is a new design and will require writing Ada objects and a new PSDL specification. The α sites at MITRE and CECOM will be examining other applications as well.

2.4.4 Problems in CAPS

Over a period of five months, a variety of features, limitations, and problems were observed in installing, using, and exporting CAPS to various locations. These locations include ISTI and MU as well as the α sites at MITRE and CECOM. These observations coupled with the studies and prior experience of Drs. Berzins and Luqi form the basis of the technical modification discussed in Section 3.2.

2.4.4.1 Features. Much of the hands-on use and evaluation of CAPS at MU was performed by masters program graduate students in the software engineering department. These students had never seen CAPS before and, although they were proficient in C and C++, had only limited experience with Ada. As a result of this work, the following features were observed:

- With virtually no help from NPS and only a brief written tutorial, the system was installed and made operational.
- Two new applications were successfully built using the PSDL, correctly compiled into Ada, and successfully run.
- Reuse of existing software atomic objects, developed in other applications and built by NPS, were used in these new applications (a flight controller and logic adder were developed). This was done in spite of the fact that the graduate students had little knowledge of Ada.
- As a result of MU using CAPS, a user's manual was developed for α site participants and a course of instruction was presented to potential users on 17 May. This is discussed in Section 2.4.7.
- MU faculty members involved with the effort now realize that CAPS is a valuable teaching tool in software engineering principles. CAPS will be incorporated into the MU graduate curriculum. The academic customer represents another potential market for the CAPS tool.

2.4.4.2 Limitations and Problems. One of the key objectives of Phase I was to identify the maturity of the CAPS tool and to incorporate the discovered problem areas into the Phase II plan. Table 2 summarizes the discovered faults and gives a few examples of each.

Our overall assessment of CAPS is that it is fundamentally sound and possesses the capabilities advertised by NPS in its publications. However, there are many improvements needed, both required and desired. We feel that the risk associated with advancing

Table 2. Limitation and Problems with the Current CAPS Version.

Limitations and Problems	Examples (Some)
CAPS installation procedure	♦ Needs to run in environments other than Berkeley 4.1.3 and 1.1.1 of Sun Ada. Enhanced flexibility is required. ♦ A tape drive should be part of the platform.
Man machine interface	♦ Needs more on-line help built into the CAPS. ♦ Needs to enhance the ease of accessing, browsing, and editing the PSDL. ♦ Needs multi-user access to the Ada compiler.
Enhance extensibility and flexibility	♦ Needs flexibility for C and C++ compilers in addition to Ada compiler. ♦ Needs to run on multiple platforms, e.g., SGI, PC. ♦ Should run with TAE as well as TAE+. TAE is free, and therefore provides cost savings.
Needs to improve basic function (e.g., browsing, debugging, editing, storing, and compiling)	♦ On-line help needs to lead operator through these functions. ♦ Storage operation not consistent. ♦ Mouse is too sensitive. ♦ System occasionally freezes. ♦ All special words need to be defined.
Desired new features	♦ Add text comment capability to graphic editor. ♦ More use of color to designate character of PSDL objects. ♦ Move most of PSDL from text to graphic.
Expand library of reusable elements, particularly for commercial application	♦ All atomic objects are Ada. Need to create these elements in C and C++. ♦ Treat examples on process control and resource management and not just command and control. ♦ Work more examples and embed into library.

to Phase II to achieve a commercial product in two years is low. This depends on the number of features that will be incorporated into the initial releases of the tool. The ISTI team has the experience needed to complete the development of CAPS.

2.4.5 Restructuring Plan

The purpose of the restructuring plan is to describe an evolutionary process starting with CAPS as we know it today to a series of commercial releases in a 1997 time frame. (A detailed plan for implementing this restructuring is discussed in the Section 3 of this report.) The current version of CAPS at the NPS, known as the NPS Research Version, and how it addresses the requirements discussed in Section 2.4.1 and outlined in Table 1, is shown in Table 3. Table 4 then shows the restructuring plan for transitioning the current CAPS version to later versions in the near-term (1 year), mid-term (2 years), and long-term (greater than 2 years) time frames.

Table 3. Status of the Current NPS Research Version.

	Status of the NPS Research Version
1. Environment supports configuration management and change control.	Partially implemented in general code. All features tested as research projects, some of which generate single purpose code.
2. Environment supports project management.	Partially implemented in general code. All features tested as research projects, some of which generate single purpose code.
3. Maintenance of design and code in same interactive environment with a shared repository.	Partially implemented in general code. All features tested as research projects, some of which generate single purpose code.
4. Human factors consideration.	Some factors well coded, but limited on-line help and some manual processes could be automated.
5. Supports real-time applications.	Excellent support through CAPS tools such as scheduler and man-in-the-loop simulation capability.
6. Automatic code generation from a graphical design.	Good capability in automatic generation of prototype code. Complete product development in conceptual stage only. Produces only Ada code.
7. Facilities to characterize and reuse software modules.	Limited library of reusable modules.
8. Integration with other software packages.	Very limited number of tools and specific versions only.
9. Prototyping and testing support.	Works very well. Some refinements in concept stage.
10. Flexibility and extensibility.	Extremely limited in classic sense. Very fluid in incorporating changes as part of new research.

Table 4. CAPS Restructuring Pan.

Time / System Features	Near Term (1995/96)	Mid Term (1996/97)	1998 and Beyond
Rapid Prototyping Tools	• β version in test. • Requires some manual scripts to link compiled Ada and TAE+ graphics.	• Provides a complete software prototyping tool. • PSDL specifications will produce Ada and C code that requires some tailoring.	• Provides a complete software prototyping tool with PSDL description in and a 4th generation language out.
Life Cycle Management Tools	• Prototype only - not ready for release.	• Management, design, implementation, testing tools and configuration management functions in β testing.	• Supports all phases of software life cycle management. • Provides the basis of management, design, implementation, testing tools and configuration management functions.
Man-Machine Interface Enhancement	• User needs at least a week to train on tool. • Some menu guidance. • Special knowledge required to use.	• Enhanced user interface. • Learned within a few days of use. • Only limited special knowledge is required.	• User is trained within hours of use, with on-line help and simple menus. • No special training for use except for high level architecture.
Modal Library for Reuse	• Limited applications library with primarily military applications. • Only Ada atomic objects.	• Extended applications library with at least 15 examples - 1/2 military, 1/2 commercial. • Some C++ and Ada atomic objects.	• Very large library of applications with both Ada and C++ atomic objects. • Few new objects needed in design.
Total Tool Integration	• Tools exist as separate non integrated modules.	• Prototype tools are fully integrated. • Initial management tools are available and integrated.	• All tools are fully integrated in a cradle-to-grave software development tool suite.
Flexibility and Extensibility	• Works only with SunOS Berkeley 4.1.3 and Ada 1.1.1. • Runs on Sun platform only. • Will produce only Ada code in prototype.	• Works with near term environment plus Sun Solaris with GNU and Sun 2.1 Ada compilers. • Also linked to Sun C++ and C compilers.	• Runs on Sun, HP, PC and SGI terminals. • Compatible with the latest version of operating system and graphic editors. • Will produce Ada, C and C++ rapid prototypes.

In general, all the system features shown in Table 4 are available in the CAPS NPS Research Version and other versions residing at various sites previously discussed. No major changes in the tool architecture are required. However, a series of incremental improvements, as shown in Table 4, will occur. In addition, the system will be placed under controlled configuration management needed to support new implementations and acceptance testing. The addition of C and C++ compilers, the application to industrial commercial applications, enhancements to the man-machine interface, and formal integration of all tools highlight the evolutionary process.

2.4.6 Source Code Review

In addition to the operational evaluation of CAPS, discussed in Section 2.4.4, a detailed code review was conducted. This review has lead to a series of recommendations involving enhancements to the source code. The code was found to be fundamentally sound but requiring a series of fixes. Some of these are listed below.

- The code needs additional commenting.
- Additional data types are needed to complete the PSDL implementation language.
- The editor must enforce error repair before proceeding with additional corrections. This feature needs to be added.
- The tool should permit the ability to add manual commands for new unanticipated applications beyond the tool's current capability.
- When compiling a new application, the system also recompiles (old) applications. This is a waste of time and processing power.

All these enhancements will be corrected in the course of evolving from the current version to later CAPS versions (see Phase II plan for details).

2.4.7 Development of a Preliminary User's Manual and Training Material for α Site Users

On 17 May 1995, MU held a one-day training course for employees of MITRE and CECOM who will be using CAPS at their α sites. The course consisted of a half day of classroom instruction, followed by a half day of hands-on training on the installation and use of CAPS. Each student was provided with a user's manual. In addition, the software environment required to support CAPS was defined in detail. The α site users who did not own the needed software to run CAPS (e.g., Berkeley 4.1.3, Ada compiler 1.1.1, X-Windows Motif, and TAE+) were able to dial directly into MU's SunSPARC terminal and access CAPS directly. A summary of the material presented during the training follows.

CAPS TRAINING OUTLINE (AM)

- Introduction/History of CAPS
- Software Prototyping (Types, Approaches)
- CAPS as a Tool for the Prototyping of Time-Critical Functions, Hard/Real Time Scheduling Introduction, and Implementation
- Application System Modeling. Implementation
- CAPS Main Components and How They Interact
- Current State and Future Developments of CAPS

CAPS TRAINING OUTLINE (PM)

- Installation/ CAPS Access
- Getting Started
- Operational View of CAPS
  - Introduction to Invoking CAPS
  - Description of Menu Features
  - Examining the Temperature Controller Graph
  - Building a Model Prototype (Except for Ada Modules)
- Demonstration of the Autopilot Model

CAPS TRAINING OUTLINE (BUILDING A PROTOTYPE)

- Entering the Graph Editor
- Adding/Deleting Operations and Streams
- Naming Operators
- Adding Color
- Returning to PSDL Editor
- Setting Stream Variable Types
- Adding Cyclical Variables
- Marking Implementation of Operators
- Setting Period Constraints
- Translating
- Scheduling
- Compiling
- Running the Prototype

A summary of the elements in the user's manual are outlined below. The most essential parts are the installation manual and a CAPS tutorial prepared by the NPS.

OUTLINE OF TABLE OF CONTENTS OF THE USER'S MANUAL

- Overview
- Site Requirements
- CAPS Installation Manual
- CAPS Tutorial
- References
- Case Studies/Sample Prototypes

2.4.8 Preparation of a Plan for Commercialization

The commercialization plan consists of four related sections. These include a technical plan, managerial plan, marketing plan, and a product support plan. An outline of the plans are presented below.

2.4.8.1 Technical Commercialization Plan. The technical plan is presented in three parts. In part 1, a series of new version releases will create a progression from the CAPS Version 1.0 software, now installed at ISTI and MU to a fully commercialized version meeting most of the refinements shown in Table 1. CAPS Version 1.0 is a subset of the NPS Research Version. This progression is shown in Table 5 and the plan itself is in section 3.2. The objective requirements are shown on the left and added capability associated with each version appears in the columns to the right. Version 2.1 will be available by October 1997. The detailed plan will take CAPS to Version 2.3, however work on Versions 2.2 and 2.3 will only be started in the Phase II STTR and will be completed later with private funds. In Section 3.3, the plan to extend the tool to Versions 3.0 and 4.0 is discussed in general terms.

During the second part of the technical plan, ISTI will enhance its set of tools and procedures required to fully support this CAPS evolution. Table 6 outlines the key items associated with software evolution. Most of the tools used in the life cycle process have been or are being acquired as commercial-off-the-shelf (COTS) products. The procedures used have been developed at ISTI. The training has been done in part at MU and will continue during Phase II.

Part 3 of the technical plan consists of utilizing the α/β sites to interactively test the system. A layout of the evolutionary process flow and hosts employed for testing at ISTI are shown in Figure 4. The use of quality gates between the various stages of development assure adequate review prior to advancing to the next stage of development. The α/β release and acceptance test hosts are SunSPARC 2 machines. The system integration will be performed on a SunSPARC 20. A variant of this configuration is also located at MU.

Table 5. Technical Commercialization Plan.

Date expected:	present	1/97	3/97	6/97	10/97
Requirements	CAPS 1.0	CAPS 1.5	CAPS 1.6	CAPS 2.0	CAPS 2.1
1. Configuration management	Treated in Version 3.0				
2. Project management	Treated in Version 3.0				
3. Shared repository	Treated in Version 3.0				
4. Human factors		On-line help improved		TAE+ script files automated	
5. Real-time	CAPS scheduler				
6. Automatic code generation	PSDL language, with partial implementation			PSDL language completed	
7. Software reuse	Treated in Version 2.2	Completion of STTR			
8. Other software integration	Some integration		Additional text editors supported. TAE supported	Other Ada compilers supported	C and C++ compilers supported
9. Prototyping, testing	Prototyping support. Simple Hardware Model for testing				
10. Flexibility, extensibility		Faults in the GUI interface		Support for versions of SunOS and Solaris	

Table 6. Life Cycle Phases.

Elements of Phases	Requirements Definition & Analysis	Design	Developments & Unit Test	Integration & Testing	Acceptance Test	Configuration Management
Standards & Guidelines	ISTI comp stds Mil Std 498 Mil Std 2167A	ISTI comp stds Mil Std 498 Mil Std 2167A	ISTI comp stds Mil Std 498 Mil Std 2167A	ISTI comp stds Mil Std 498 Mil Std 2167A	ISTI comp stds Mil Std 498 Mil Std 2167A	Mil-Std-483
Procedures	Waterfall/Spiral Software Cycle					
Tools	RDD-100	RDD-100	ISTI Development Platform	ISTI Integrated Platform	SCCS & SRC	SCCS & SRC
Quality Assurance	Use of TQM coupled to quality gate 1	Use of TQM coupled to quality gate 2	Use of TQM coupled to quality gate 3	Use of TQM coupled to quality gate 4	Use of TQM coupled to quality gate 5	Use of TQM coupled to quality gate 6
Training	Staff at MU and consultants	Staff at MU and consultants	Staff at ISTI and MU	Staff at ISTI and MU	Staff at ISTI and MU	Staff at ISTI

2.4.8.2 Management Plan. The thrust of the management plan is outlined below. Key elements of the plan include the following:

- Special emphasis will be placed on non-government application and sales.
- The focus of configuration management, documentation, software development, and product support will be conducted at ISTI and by the consultants.
- The focus of application development and acceptance testing will be performed at MU and Monterey CA, where the consultants, Drs. Berzins and Luqi are located. Integration will be performed at ISTI.

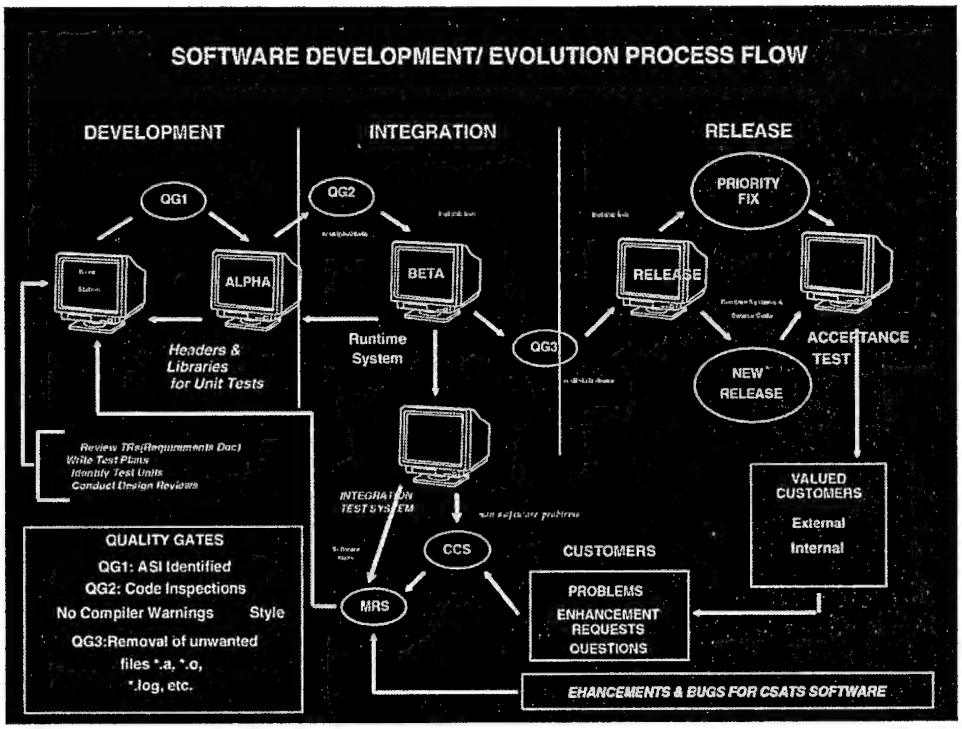


Figure 4. ISTI Computer Hosts.

- β site testing will be conducted at several locations. This includes MITRE and CECOM for government applications, AT&T Bell Laboratories, and Telos for domestic applications, and MU and Georgia Tech for university use.
- An extensive commitment will be made on the part of ISTI and MU involving their own funding, personnel support, and equipment use. These funds are in excess of the dollars funded by the current STTR Phase I effort. This involves approximately $100K from ISTI and $50K from MU. This represents a 30-percent augmentation to the STTR Phase II effort.

2.4.8.3 Product Support Plan. Product support will be offered in several forms. Following is a summary.

- On-line help will be built into the CAPS software.
- A telephone hot-line will be offered at ISTI and MU.

- The product will be released with supporting documentation, including a user's manual, training manual, and related specification sheets.
- Full configuration management will be available at ISTI. Any confirmed faults will be incorporated into new releases and the customer will be offered telephone and written communication connections. This information will also be included in new releases.

2.4.9 Proof of Concept Demonstration

A proof of concept (POC) demonstration was developed at MU. The demonstration illustrated the ability of a new CAPS user to understand the features of CAPS and create a CAPS prototype. Several new prototypes were developed at the MU α test site. The MU laboratory configuration is shown in Figure 5.

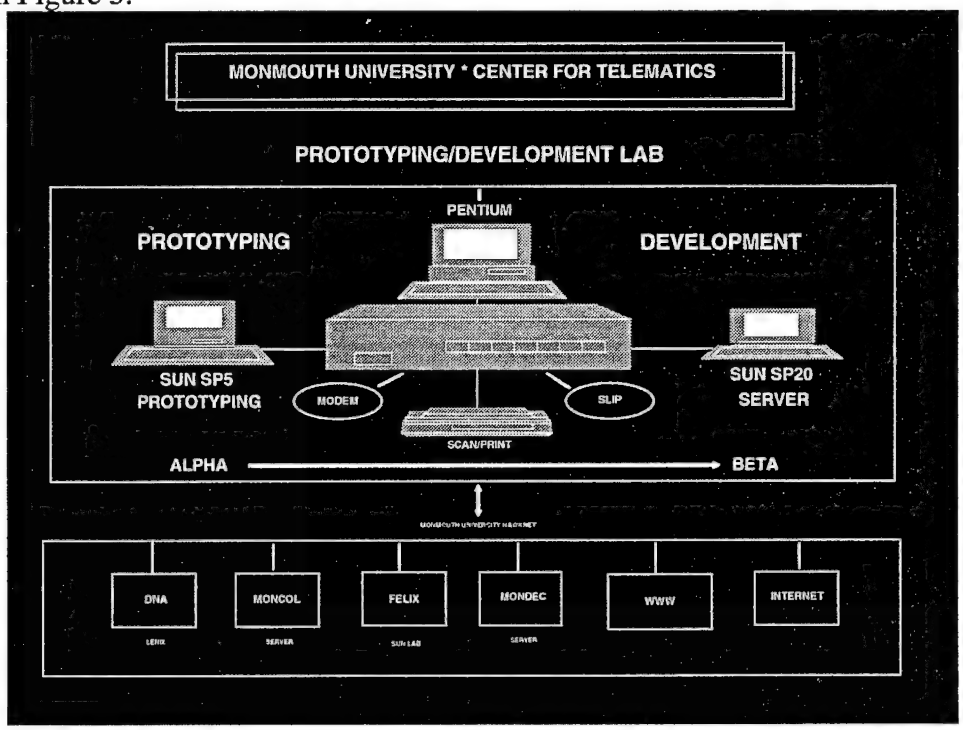


Figure 5. Monmouth University, Center for Telematics, Prototyping/Development Lab.

2.4.9.1 POC Objective. CAPS can be used as a first step in designing the software needed to run a real time system. To demonstrate this fact, several prototypes were created in the Monmouth University's CAPS environment.

2.4.9.2 Examples. Two of the major prototypes developed were the Weapons-Command Center (WCC), which dealt with the detection, assimilation, and reaction towards friend and foe aircraft, and the Weighing Station Traffic Controller (WSTC), which handled the routing of trucks through a weighing station on any highway. The WSTC will be discussed here.

Taken from the real-life situation where trucks traveling on a highway are periodically weighed and charged, the WSTC attempts to duplicate the processes to highlight a few of the CAPS

capabilities. Periodically, trucks climb the ramp off a highway towards the station. Entering a queue, they wait until the weigher is empty. A traffic signal flashes green when the next truck in the queue is free to move onto the weigher, and flashes red otherwise. After a specified period on the weigher, the trucks climb down the ramp back onto the highway, resuming their journey. When there are no trucks in either the queue or on the weighing station, the traffic signal switches periodically between red to green, until a time that a truck does enter the queue. The software resulting from this prototype can be used to examine the requirements of the sequence and can eventually become the Ada code needed to design the system controller.

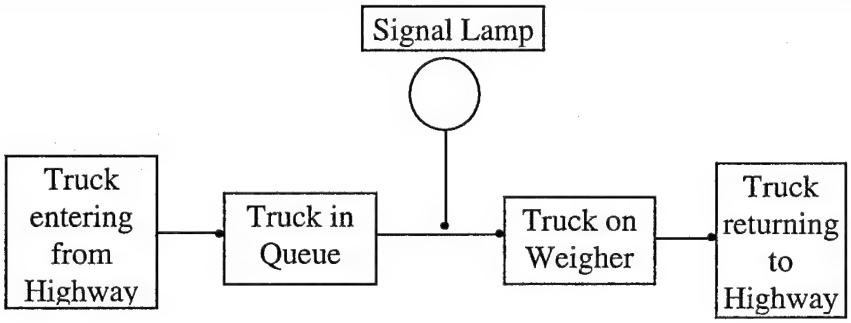


Figure 6. Symbolic Illustration of WSTC Problem.

2.4.9.3 WSTC Approach. Solving the WSTC problem is relatively simple using the CAPS software. The problem can be described in the following Exclusive OR scenario:

Truck present in Queue	Truck present on Weigher	Traffic Signal
No	No	Switch between Signals
No	Yes	Green
Yes	No	Green
Yes	Yes	Red

Starting off the prototyping process, one first creates atomic operators and streams in the graphic editor to automatically write the PSDL required. Following the attaching of proper timing attributes and conditions to each operator and stream, one must write the Ada code for each operator for compilation to be successful.

A minimum of five operators were needed to create the prototype - a generator (Onramp), a queue (Queue), a weighing post (Weigher), a traffic signal post (Lamp), and one to return the trucks to the highway (Offramp). The Lamp would need to know the status of both the Queue and the Weigher before making its decision to flash the correct signal. The action of the truck in the queue is triggered by the traffic signal that is generated at the Lamp (if red, the truck is halted, and if green, the truck moves onto the weigher). Hence, one notes that the queue operator is sporadic, while all other operators are periodic.

2.4.9.4 Experience with the System

I. If a prototype has to be developed, an important consideration is the construction of the PSDL. If the logic of the system under development is inaccurately reflected in the prototype, a typical result is to have no schedule system. This is the experience with the WSTC prototype.

II. In the current implementation of CAPS, the context switching time of the schedule (on the SUN platform) seems to be around 20 ms. Maximum execution times (METs) close to 20 ms seem to generate instabilities in the scheduling. In order to improve the schedule, we suggest the development of a benchmark to measure the parameters of the schedule. This benchmark can lead to conclusions associated with improvements to the CAPS schedule.

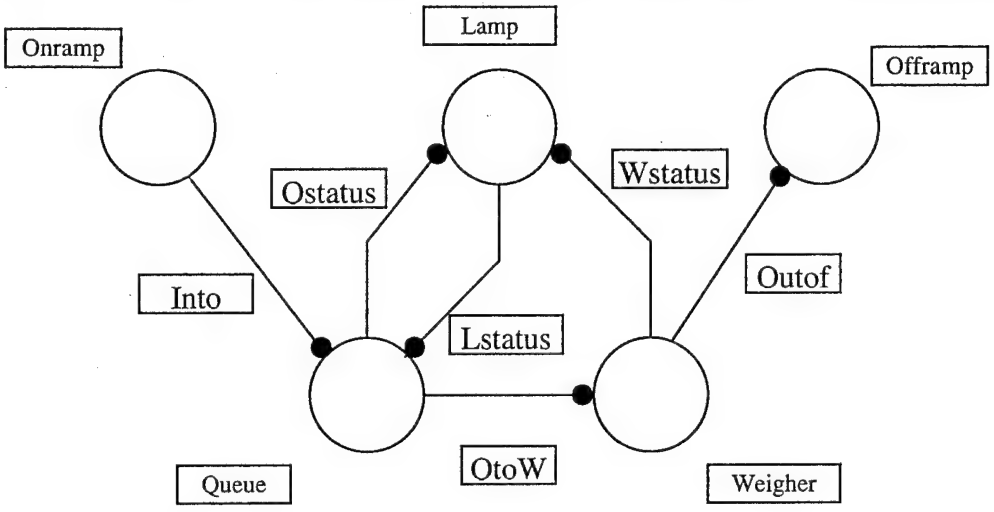


Figure 7. Dataflow Diagram for WSTC Prototype.

2.4.9.5 Results. The scheduling shown below shows the times that each operator runs in the appropriate sequence, with the number of instances that it fires in the scheduled length.

Load Factor 0.350
Schedule Length 800

Operator	Start Time	Stop Time	Instance
WSTC	0	0	1
Weighter	0	50	1
Lamp	50	100	1
Onramp	100	150	1
Offramp	150	200	1
Queue	200	250	1
Lamp	450	500	2

Sample output displayed after the Ada source code for each of the operators was compiled, and executed :

- Truck rumbles up the ramp into the queue.
- Truck drives out of the queue.
- Green Light.

SENSOR UPDATE: Truck on the weigher.

- Red Light.
- Truck rumbles down the ramp.
- Green Light.

SENSOR UPDATE: No truck on weigher.

- Automatic Blink : Red Light.
- Truck rumples up the ramp into the queue.
- Green Light.

SENSOR UPDATE: No truck on weigher.

- Green Light.
- Truck drives out of the queue.
- Green Light.

SENSOR UPDATE: Truck on the weigher.

2.4.10 Summary of Phase I Results

The set of objectives for Phase I of this effort has been met. A set of requirements has been defined and a related commercialization plan developed. CAPS has been successfully exported to ISTI, MU, MITRE, and CECOM. As a result of α site testing, a set of limitations and problems were defined that will be resolved in Phase II. Also, a preliminary user's manual and α site training was completed. A preliminary list of prospective customers has been developed. During the last week of June 1995, a "proof of concept" demonstration was conducted at MU. The enthusiastic acceptance of CAPS and the lack of fundamental problems in the software indicates that a commercial product can be achieved in a two-year time frame. However, due to the need to fully debug, test and configuration manage the product, the need for STTR funding of Phase II is essential.

SECTION 3

CAPS REENGINEERING PLAN

3.1 Introduction

The CAPS reengineering plan was created by our consultants in Monterey after consultation with ISTI. The elements of the future product line were decided upon first and the path to the products was then mapped out. ISTI and the consultants balanced the features that we wanted to see in the most advanced CAPS product at the end of the two years against the projected manpower for the reengineering effort. Section 3.1.1 describes the product line that should be attainable during Phase II and two more additions to the Version 2.0 product line that will be introduced after the end of the STTR Phase II. Section 3.2 contains the plan written by our consultants and it has two subsections. Section 3.2.1 describes the code restructuring and section 3.2.2 describes the creation of the user documentation that will accompany the commercial software. Finally, for completeness sake, section 3.3 describes future version of CAPS that we envision.

3.1.1 ISTI Commercial CAPS Product Line

A sequence of products illustrated in Table 5 will be developed in Phase II. In addition, a projected set of products shown in Table 8 continues this release process through 1999. These releases will be funded by ISTI. The first release, Version 1.6, will contain a number of rapid prototyping features. These include an automated real-time scheduler, enhanced on-line help, PSDL language for system specification, automated prototype testing, and enhanced GUI interface. CAPS Version 1.6 will be an inexpensive product aimed at companies that have only limited programming resources.

We will market a product at the end of Phase II (Version 2.1), which will incorporate four of the five elements mentioned in Appendix A. See Appendix A for a description of all the features of CAPS such as the Evolution Control System (ECS), the database of reusable code segments, etc. After the STTR Phase II is complete, CAPS Versions 2.2 and 2.3 will be finished using private funds.

We will discuss the changes required to transition CAPS Version 1.0 to produce CAPS Versions 1.6 and 2.1 with the aid of Table 5. The following is a detailed description of the features that need changing and how the change will affect the marketability of CAPS. Table 5 shows how the various planned stages of CAPS relates to the requirements outlined in Table 1 (located in Section 1.4.1). Version 1.6 is expected to end the Version 1.0 product line. The last of the Version 2.0 product line will be Version 2.3, which will be completed after the STTR Phase II effort. The final products are compared in Table 7. Then work will begin on the Version 3.0 line, which will focus on a software development management tool.

Table 7. Environment and Features of CAPS Versions 1.0, 1.6, and 2.3.

ITEM	*CAPS 1.0*	*CAPS 1.6*	*CAPS 2.3*	*Essential?*
1	Berkeley UNIX SunOS only	same as 2.0	Berkeley or AT&T UNIX SunOS or Solaris	Yes
2	Sun Ada Compiler	no change from 1.0	All popular Ada compilers	Yes
3	Ada only	no change from 1.0	Ada, C and C++ C and C++ have limited implementation	No
4	VADSedit, vi, emacs	no change from 1.0	open architecture	Yes
5	TAE+	TAE	TAE, TAE+, perhaps others	No
6	TAE+ script file	no change from 1.0	enhanced automated script file	Yes
7	No software database (ONTOS used at NPS)	no change from 1.0	software database ONTOS or another choice	No
8	Simple Hardware Model	no change from 1.0	Enhanced Hardware Model	No
9	Single user	no change from 1.0	Network	No
10	Limited User Friendliness	same as 2.0	Enhanced User Friendliness	Yes
11	Manual installation	same as 2.0	Automatic installation	Yes

3.1.2 Explanation of Features

1. Running CAPS Under Multiple OS. The CAPS libraries have been compiled at the NPS using the SunOS operating system, which uses Berkeley UNIX. One of the installation problems uncovered was that these libraries would not work with Ada code created by the Sun Ada compiler for the man-in-the-loop simulation. The problem occurred because the software is running

under Solaris and the Solaris operating system uses AT&T UNIX. The immediate solution was to use the SunOS operating system, but many users will want the option of using Solaris or SunOS. The commercial product will have two sets of libraries and the installation procedure described in item #11 of this section will ask the user which operating system will be used. The correct set of libraries will be installed. This feature is essential for the CAPS 2.1 product.

2. Expanding the Ada Compiler Family. Presently, the CAPS 1.0 system can only interface with the Version 1.1 of the Sun Ada compiler and no other compiler. CAPS can find better commercial acceptance if developers have greater flexibility in selecting Ada compilers. Work was under way under in Phase I to allow the GNU Ada compiler to be used. The GNU compiler was an obvious choice as a compiler, because it is a public domain compiler and is free. The GNU compiler will appeal to purchasers of CAPS Version 1.6. CAPS 2.0 will have a list of supported Ada compilers. The installation program will present the list to the user and create script files appropriate for the compiler chosen. Only the script files will be affected.

3. Extending Code to C and C++. Real-time military systems are required to be written in Ada and the CAPS code was designed for Ada. Many real-time civilian systems are written in C++ and there is a general aversion to Ada in the civilian community in the United States. This is not true in Europe. There is no fundamental reason why CAPS can not be used to create C or C++ coding and it should be simple to allow the user to program in C or C++. C++ is frequently used and if the work to make C++ available in CAPS is completed, then very little additional work would be needed to make C available also. This feature is highly desirable in CAPS Version 2.1.

4. Adding UNIX Editors to CAPS. CAPS has integrated VADSedit into its script files. The current CAPS version incorporates VADSedit, emacs, and vi editors. We will incorporate additional editors besides the VADSedit, emacs, and vi UNIX editors.

5. Adding TAE and Other Graphical Tools. The use of TAE+ makes producing the man-in-the-loop simulation much easier for the programmer. TAE+ helps generates the required Ada code (or C or C++ code) reducing the programmer's workload. The programmer can use the TAE+ interactive Workbench screen to specify the GUI attributes without programming knowledge. The use of TAE+ is optional in CAPS, as the programmer can choose to create the needed GUI. The original TAE produced was developed by NASA and is in the public domain. TAE+ is a commercial product with a richer set of features not found in TAE. Nonetheless, it will be very simple to make TAE work in CAPS. We will survey the field of graphical user interface builders to determine what others we could allow, and how much additional work would be required to include them. Including a TAE option into CAPS is desirable but not essential.

6. Automation of TAE+ Scripts. At the present time TAE+ is integrated into CAPS, but its use is not fully automated. The programmer must spend significant time reworking the TAE+ output, before it can be used with CAPS. Automating the TAE+ script files is essential for a successful CAPS Version 2.0 product.

7. Automating the Database Search Procedure. The database of reusable code segments at the Naval Postgraduate School works very slowly. The database uses ONTOS, which is an objected oriented database (OODB) manager. The problems using ONTOS are well known to the ISTI team. Currently, each transaction requires opening and closing of the OODB. Serious consideration will be given to all of the commercially available OODB managers. This feature will be added to Version 2.2 after the completion of the STTR Phase II effort.

8. Improvement of the Hardware Model. CAPS checks the timing constraints used in the PSDL by two methods. First, it checks the timing constraints written in the PSDL with the Scheduler. Then it is further checked when the CPU and operating system are performing the man-in-the-loop simulation with the aid of the Hardware Model.

The present CAPS Hardware Model allows the user to input the ratio of the "CPU speed" of the target system to the "CPU speed" of his workstation. In Version 2.3, it will assign portions of the prototype to separate CPUs. This will enhance realism.

9. Adding a Multi-user Environment to CAPS. CAPS Version 1.0 runs on a single workstation and there is no method for sharing a prototype across a network of workstations. In Version 2.1, CAPS will allow multiple programmers to work on the same prototype. A distributed capability will be added to versions beyond the current STTR versions.

10. Enhancing User Friendliness. Three primary areas will be improved. These include (a) more on-line help, (b) enhancing ease of accessing browsing and editing the PSDL, and (c) reducing manual scripts to complete finished code. This work entails direct additions to the current code.

11. Automation of Startup and Installation Procedure. The installation of CAPS Version 1.0 at ISTI and MU was manpower intensive. Version 1.6 will quiz the user as to the compiler and operating system being used and automatically create the script that will install elements needed for start up. This feature is essential to CAPS Versions 1.6 and 2.1, because the work required to install the CAPS Version 1.0 program is sufficient to discourage sales.

3.2 The Reengineering Plan

The Phase II effort will correct perceived problems in Version 1.0 of CAPS and make a limited number of enhancements designed to make the current core capabilities of CAPS more usable by customers. The scope of the proposed activities has been determined using the criterion of making CAPS as useful as possible to prospective customers operating within the limits of available resources with which to realize enhancements. The following plan is the result of a cost/benefit analysis carried out during Phase I, with the objective of producing the best possible product at the end of Phase II, consistent with cost constraints and the realities of the software development process.

There are two different aspects to this effort, the software and the documentation.

Software upgrades will be the responsibility of ISTI, with guidance and assistance from the consultants. The bulk of the work will be done by two professional programmers employed by ISTI. Due to the substantial complexity of the CAPS design and the theoretical advances embedded in the software, a much larger learning effort is expected than is common for most software development projects. Since it is anticipated that frequent communication between the programmers, ISTI, and the consultants will be required, it is considered essential that both programmers be situated in California, both at the same site. Availability of office space and equipment suggest this site should be at ISTI, although some time may be needed in Monterey, especially at the beginning of the project when there will be a substantial learning effort by the programmers to understand the system. It may be possible to reduce this need by hiring former NPS students who participated in the initial development of CAPS.

Testing, evaluation of usability, producing documentation, and producing educational materials will be the responsibility of Monmouth College, with review by ISTI and the consultants. Different aspects of this effort will be carried out by the staff and by students. This will provide the benefits of an independent testing organization, as well as greater effectiveness at detecting and correcting faults of omission in the documentation, because the testers will be somewhat removed from the development effort. This should help expose faults regarding missing information in the documentation.

The reengineering and repair process will be done in stages, producing intermediate versions that will be tested and evaluated with respect to both the current state of the user documentation and actual trial use while the next enhancement is being developed. The evaluation process will be integrated with an ongoing activity aimed at completing and refining the documentation, to ensure that the documentation contains all of the information needed to effectively use the system. To this end, problems perceived by testers and users of the intermediate versions will be recorded and tracked to the parts of the documentation responding to the problems, as well as to any software enhancements that are developed in response to the problems. This record will be systematically reviewed to ensure that all concerns have been addressed.

3.2.1 Software Reengineering

(1) Repair known faults (Version 1.1). Trial use during Phase I has resulted in a set of problem reports. These reports will be organized and used as the basis for an initial set of corrective actions. Known faults impact many parts of the system, although most of the faults reported in the early stages of trial use are most closely related to the editors and user interface.

(2) Test the system and documents to find currently unknown faults (all versions). This process will continue throughout the project, based on the most recent version delivered to Monmouth College. Results will be organized and fed back to the developers and/or to the documentation effort for corrective action.

(3) Modify the editors to prevent data loss without warning (Version 1.2). A currently known problem with the CAPS syntax directed editor is that after detection and announcement of a syn-

tax error, the system allows the user to continue editing without correcting the syntax error. If the user does this and later saves the design, an empty file results without any further warnings. This is considered to be a serious flaw because beginning users will likely experience the fault and lose all of their work, resulting in extreme dissatisfaction with the product. This may impact generated code, and is likely to be technically challenging.

(4) Remove unimplemented features from the user interface (Version 1.3). The current version, CAPS Version 1.0, is a research prototype that has features in the interface that are not implemented and that will not be included in the products resulting from Phase II. These are placeholders for the results of future research. They should be removed from the product version to avoid confusing users - everything visible in the delivered product (Version 2.0) should work reliably. This effort impacts mainly the syntax-directed editor and the main CAPS user interface.

(5) Add commands for manually creating new versions and for deleting versions and entire prototypes (Version 1.4). These features are missing from the current interface because they are expected to be automated in future versions of CAPS, without need for any explicit actions from the users. Since these automated facilities will not be included in Version 2.0 and customers will need to create and manage multiple versions of their prototypes, the interface must be extended with facilities to perform these operations manually. This affects the main user interface and shell scripts, and requires the creation of some new software modules.

(6) Repair faults found during testing (Version 1.5 - result is the minimal entry-level product). The plan calls for developing two versions of CAPS, a low-cost entry level version and a serious production version. The entry-level version is intended as an educational tool and as a means for organizations to acquire some of the basic CAPS capabilities without incurring a great deal of risk and expense. It is intended that trial use of the entry-level version should enable pilot projects to demonstrate the benefits of using CAPS and pave the way for more extensive acquisition and use of the serious production version. Version 1.5 is considered to be the simplest subset of CAPS that is viable as a product. The entry level product must operate reliably in order to attract customers to the serious production version. The scope of this task is the entire system, and its magnitude is unknown until completion of Step 2.

(7) Add options to allow use of other text editors (Version 1.6). The current version of CAPS uses the UNIX vi editor for all text objects. This is a useful default because vi is distributed along with the UNIX operating system, and will be available in all expected operating environments. Programmers are often strongly attracted to using the editor they are used to, and not everyone likes vi. Thus an attractive product should support interface to other text editors if they are available. One popular choice is emacs; others may be included as well, depending on feedback from the β sites. This task affects mainly the CAPS user interface and some of the shell scripts.

(8) Add options to allow use of other Ada compilers (Version 1.7). The current version of CAPS is known to work only with Sun Ada Version 1.1, which is specific to the SunOS version of UNIX. Especially since Ada compilers tend to be expensive, most organizations will be

strongly attracted to using whatever Ada compiler they already own, making it advisable to include options for interfacing to all of the most popular Ada compilers. The GNU Ada compiler (GNAT) should be included as an option because it is available at no cost. This task affects mainly the CAPS user interface and some of the shell scripts. Changes to the architecture of the generated Ada code and the corresponding program generators may be needed if the new compilers are not fully compatible with the Ada structures currently generated. This is more than a matter of using only validated Ada compilers - the original Ada architectures had to be modified to avoid bugs and limitations of the Sun Ada compiler, and there is no guarantee that this will not happen when converting to another compiler.

(9) Improve integration of CAPS with graphic user interface generator (Version 1.8). CAPS Version 1.0 is loosely integrated with the TAE graphical interface generator. The current version requires a fairly lengthy manual process to modify the output of TAE to conform to the CAPS interface conventions. This process is documented in detail and works, but is not convenient. It should be simplified and partially automated to make it easier to use. This effort requires redesign of parts of the CAPS runtime structure and creation of some new software modules.

(10) Add options to run under Solaris (Sun's new operating system), and adjust the installation and initialization procedures (Version 1.9). CAPS Version 1.0 runs on SunOS. The binary file formats and some details of the system commands in Solaris (AT&T UNIX) are different than those of SunOS (Berkeley UNIX). Since Sun is pushing to transition to Solaris from its older operating system, it should open a larger market for CAPS if CAPS can run on both operating systems. This change requires redefinition of the directory structure in the CAPS release, addition of new environment variables, and extensive changes to shell scripts and CAPS installation procedures. It may result in the discovery of additional, currently unknown operating system dependencies, since CAPS currently runs in a stable manner in only one operating environment.

(11) Complete the implementation of PSDL data types (Version 2.0 - main product). The predefined data types of PSDL, notably the generic set, sequence and mapping types, are not implemented in CAPS Version 1.0. Availability of these types will greatly extend the information modeling capabilities provided by CAPS, and should make it more attractive to users seeking to explore complex applications. This change requires creation of new modules and is likely to impact many parts of CAPS, including the PSDL translator, schedulers, editors, and other internal modules. Since the data type facility has not been thoroughly tested, repairs to currently unknown faults are likely to be needed.

3.2.2 Documentation Enhancement Activities

(1) Collect, organize, and track the status of problem reports from system testing and the β sites. This activity is essential for achieving a high quality product in the eyes of potential customers.

(2) Create a document to help new users get started. CAPS supports a software development process that is considerably different than traditional software development processes. It is

essential to provide an easy and gradual path for customers to gradually extend what they know and enhance their capabilities without requiring a huge investment of their time. Once they are comfortable with using the system and feel they can get useful work done, they will be motivated to gradually learn more to enhance their abilities and to realize the benefits of the new processes that CAPS enables. This document should therefore introduce a simple and minimal set of CAPS capabilities that is sufficient to empower customers to do simple applications.

(3) Create a reference manual. Current documents provide only an introduction to the most basic features of CAPS. A well indexed reference manual is needed to enable advanced users to get the most out of the capabilities of the system. This includes information about the process that CAPS supports as well as information about the tools.

(4) Document available tool commands. Current documents do not explain the details of all available commands, especially for the editors. People will not be able to use these features unless they know that they exist.

(5) Document error messages, explain probable causes and possible corrective actions. There is currently no written description of CAPS error messages. This is a problem, especially for new customers, who will certainly make mistakes while they are learning how to use the system. A well-indexed explanation of possible errors will add value to the product.

(6) Remove all references to manager mode and other unimplemented features from the documentation. Manager mode consists mostly of the Evolution Control System, which is a Version 3.0 feature. Experience with trial use at the β sites indicates that users get frustrated when they find features in the documentation that they want to use, and then discover that the feature is not yet implemented.

(7) Produce a user's guide for the prototyping language. Customers will have to learn PSDL to use CAPS. Although information about PSDL is available in the research literature, it is not in a form that is very readable. A user-oriented presentation of this information is needed.

(8) Complete the on-line documentation facilities for Version 2.0. CAPS Version 1.0 has a framework for on-line documentation, but this is very sparsely populated with actual information. The structure should be filled out with enough detail to answer most questions commonly posed by users. The project should record all questions actually asked by users at β sites to gauge coverage of the on-line documentation.

3.3 The CAPS Product Line After the Phase II STTR

The technology developed during this STTR Phase II effort will be sold to generate revenue and to build a base of CAPS users. We have planned to market Version 2.1 at the end of the STTR Phase II effort. There are two major upgrades expected after CAPS Version 2.1 is released, explained earlier, and then work will start on Version 3.0 and eventually Version 4.0. Table 8 continues the product line plan started in Table 5 during the STTR Phase II effort.

Table 8. Future Technical Commercialization Plan.

Date Expected:	3/98	6/98	12/98	unknown
Requirements	CAPS 2.2	CAPS 2.3	CAPS 3.0	CAPS 4.0
1. Configuration management			ECS	
2. Project management			ECS	
3. Shared repository			ECS, Merger	
4. Human factors				
5. Real-time				
6. Automatic code generation				Automated finished software production
7. Software reuse	Implemented with search based on software features		Search based on software and documentation. A populated database	
8. Other software integration				
9. Prototyping, testing		Hardware Model upgrade		Hardware Model expansion to special hardware
10. Flexibility, extensibility			Other workstation platforms	

3.4 CAPS Version 3.0

CAPS Version 3.0 is based on research completed at the NPS. Some of the elements of Version 3.0 are considered to be well coded in the NPS Research Version of CAPS and some elements have been removed from the NPS Version or exist as single purpose demonstration code, whereas general purpose code would be required for a commercial product. Because the elements of Version 3.0 are not as fully coded as the elements of Version 2.0, it will take additional work to produce the commercial version. Along with reengineering the existing code, there will be some developmental work required for currently non-existing sections.

Version 3.0 will be devoted to the computer aided evolution of software and as such, is aimed at computer aided tasks required for large projects. With Version 3.0 will come the Automated and Integrated Repository and Planning System, which is part of the Evolution Control System (ECS). The repository will store the design documents in an object oriented database using the same database manager program as the software database. The searching of reusable code segments will be more complete, because the CAPS system can use both the code itself and the design specifications to find the reusable code. At the NPS in a demonstration of this system, the list of code modules matches included a metric which showed how well the specification and coding matched the requirements. Software modules that had a metric of 100 could be used as an identical match without the programmer having even to look at the code.

The configuration management functions of the ECS freezes old versions of the design documents and coding. The design documents are stored in the database in a manner that allows the ECS to associate any design document with code modules affected by the document and vice-versa.

The ECS maintains an internal model of the skill level of all the programmers, which is updated as the programmers finish their assignments. The internal model requires that the manager assign a level of difficulty to each module and then CAPS measures the time required by the programmers to finish the modules. CAPS also maintains schedule of which programmer is assigned to which module and the dates that the modules should be started and completed. CAPS generates a non-intrusive status report daily on its belief as to the progress of the project compared with the schedule. If a module is not completed on the appointed day, CAPS issues an alert to the project manager via e-mail indicating the existing schedule is no longer valid. CAPS contains automated rescheduling tools that allow the manager to find the best realistic schedule to finish the project. If work is completed faster than required, the same process with alerts and rescheduling occurs.

Because CAPS has the project's document trail with forward and backward traceability, CAPS can perform automatic briefings to programmers. When the programmer indicates to CAPS that he has finished a module, CAPS refers to the workplan to determine the programmer's next task and finds all the relevant documents in the repository and places them in the programmer's disk area along with an e-mail message. The programmer can then read the documents on his own without any action from the manager. The proper software items will also be placed in his disk space. Meanwhile, CAPS automatically forwards the finished module and related documentation to the manager with an e-mail indicating that they are complete. The manager can review the software and documentation and determine if the programmer was successful. Once the manager is satisfied, CAPS places the new module and documents into the repository. They are frozen and can only be updated when the version number is incremented.

Finally, CAPS Version 3.0 will contain the Merger facility. This facility allows a design to branch off into two separate designs and then become reconnected again at a later date. Consider the following case, which is typical for all software development. A finished software product is released to the public as Version 3.0. After the release of this product, work is started on additional features that are intended for a Version 4.0. Also, after release of the product, errors and poor design features are reported by the customers of Version 3.0. These errors are fixed as Versions 3.1, 3.2, etc. When Version 4.0 is finished, there is the task of tracking down those features that are common to Version 3.0 and Version 4.0, which were fixed in later releases of Version 3.0. CAPS does this task automatically with the Merger facility. All of the common features of Versions 3.0 and 4.0 that have been changed in Versions 3.1 and 3.2, can be incorporated into Version 4.0. For those changes that CAPS can not find with certainty, a diagnostic message is produced.

Also, in the CAPS Version 3.0 effort we will expand the CAPS workstation operating environment to non-Sun workstations, such as SGI workstations and others.

3.5 CAPS Version 4.0

CAPS Version 4.0 is purely conceptual and forms the basis for future research planned by our two consultants. CAPS Version 4.0 will automate the transformation of the prototype software into a product.

The developed software is always referred to as a prototype in our work, even as the modules are coded into the finished design. The reason for using this terminology is that the finished product will be embedded into a computer system that is not be related to the workstation used for software development. There are five distinct reasons why prototype coding will have to be modified before use in the target system. First, real-time systems tend to have timing constraints that are difficult to satisfy, forcing the developer to use special purpose hardware. This hardware may have algorithms coded in hardware rather than software, such as digital signal processors. Vector or array processors can speed up the processing time for a restricted set of calculations. Although special purpose hardware can seem very expensive when compared with the general purpose processor found in a workstation, there is usually an excellent reason for using special purpose hardware in real-time systems. The target computer may be a computer similar in design to an ordinary workstation plus some special hardware. The Ada, C, or C++ performing the function of the special hardware must be either rewritten, to take advantage of the hardware, or eliminated because the function is coded in hardware.

The second reason is that real-time systems may be embedded into something like a missile seeker. In this case the target computer is not at all similar to the workstation used to design the software. In this case it is clear that the coding that CAPS writes to tie together pieces of the system was designed for a workstation, but must be replaced by other coding. The differences may not be great, but some changes will be required.

Third, the workstation environment supports the man-in-the-loop simulation. In this simulation the input is provided by the man using his mouse and keypad to start off the simulation and adjust the simulation while it is running. The output is provided by the workstation monitor with simulated output representing, as an example, a radar screen and a "God's eye view" of the battle field. The actual system will have sensors as input and special purpose hardware (i.e., a radar display) as output. The elements in the prototype that represent the input and output must be removed and replaced with input and output drivers appropriate for the target system. TAE+ and X-windows were used to create the simulation software, so the code that requires replacement was very inexpensive to produce, but this coding bears little resemblance to the required coding.

The fourth reason is that the Ada, C, or C++ code, which ties together parts of the system, could be optimized. Even if the target system is a workstation and the input/output features are the same as the simulation written by TAE+, the timing constraints may require faster processing than provided by the code generated by CAPS. The code generated by CAPS is actually very good considering that it is machine generated, but a human programmer can optimize the code in places.

The fifth and final reason is that the CAPS-generated coding will run the system on a single workstation, even though the target system may have multiple processors. The coding generated by CAPS must be broken up to separate portions, one for each processor.

There may be a requirement that the entire system be coded entirely in C or C++ instead of Ada. This will not require any special features by CAPS, because there already exists tools that will translate Ada into C or C++. The atomic modules will be in C or C++ and the CAPS generated coding can be translated automatically, if needed.

CAPS Version 4.0 research will be aimed at automating the changes required to convert the prototype into the product. The Ada, C, or C++ coding will be automatically converted into the required code once the target system is specified. Also, as a part of the process, a code optimizer will make a final pass through the coding for optimization. If the system is written in Ada, then the atomic modules could also be optimized with this tool as well.

APPENDIX A

THE CHRONIC SOFTWARE CRISIS AND CAPS

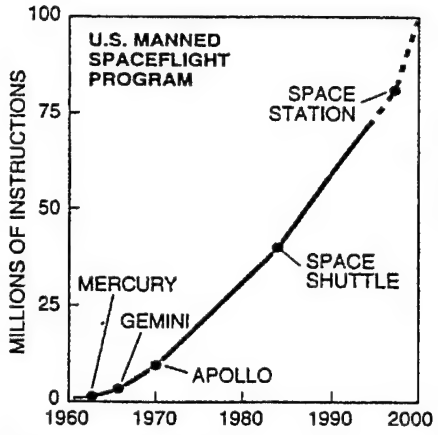
A.1 The Crisis

The Denver International Airport's (DIA) well publicized delay resulted in a direct loss of $1.1 million per day to the city and the airlines until the airport opened. The delay lasted for 16 months and the project overrun was greater than $2 billion. The baggage handling unit built by BAE Automated Systems, that has shouldered most of the blame for the delay, was initially priced at $189 million, but was finished at a cost of $600 million with a non-automated system constructed next to the automated one. The president of BAE Automated Systems, Gene Di Fonso, blamed a "rash of programming nightmares" for the system's problems. An interesting feature of this system was that luggage was placed and removed from cars, which never stopped moving during the loading and unloading. A frequently cited problem was that the cars arrived too early or too late at the loading locations and the bags that missed being placed into these cars created jams on the tracks and the bags themselves were demolished. (See References 2 and 3 for more details.)

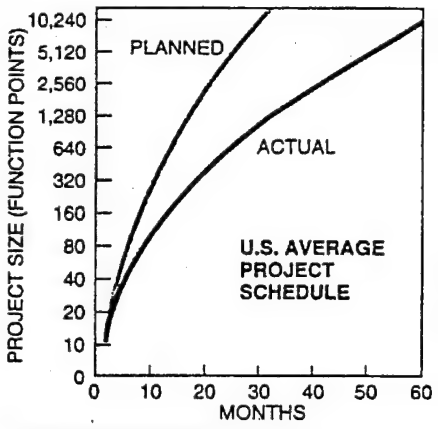
On May 7, 1994, the *Clementine,* a joint DoD and NASA deep space science probe, accidentally fired the thrusters of its Attitude Control System continuously for eleven minutes, depleting its supply of maneuvering fuel. At the time, it was being instructed to intercept the deep space asteroid Geographos. The *Clementine* had just finished a successful 70 day mission of mapping the lunar surface. A software bug was responsible. *Clementine* was built and launched at the cost of about $100 million and was considered in the space exploration community to be a modest, low-cost unit.

The Federal Aviation Administration (FAA) canceled two of the four major parts of its Advanced Automated Systems (AAS) and scaled back a third. It spent $144 million dollars on failed components before cancellation. The fourth and largest part is $1 billion over budget and is being reviewed to see if it can be salvaged. The reviewers identified poor documentation as a hindrance to their review, which has forced them to rely upon interviews with key personnel to determine the system's architecture. IBM Federal Systems did not have any Ada programmers on its staff, when it signed a contract to write one million lines of Ada code. (See References 4 and 5.)

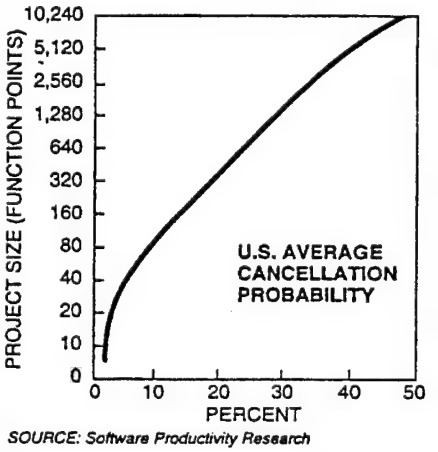
All of the above disasters have been laid on the doorstep of software developers (see Reference 6). The state of software development for any and all types of software has been considered a trouble prone area since the beginning of the computer age. Despite mankind's long involvement and understanding of the problem, the three examples given above are all current events.



SOURCE: Barry W. Boehm



SOURCE: Software Productivity Research



SOURCE: Software Productivity Research

Figure 8. As software development expands, so does its associated problems.

Also, the three examples given above are of real-time, embedded systems. In general, a system is called a "real-time system", if the system has hard timing constraints. A real-time system is one that is controlling a process and if the software makes an error or if it simply takes too much time to complete a task, control will be lost. Real-time systems are multiplying rapidly in both the civilian and military sectors.

Figure 8 contains three illustrations, which address the magnitude of the problem. The amount of software embedded in commercial products is said to be doubling every two years. The three examples given at the beginning of this section may be anecdotal, but the software crisis is universal, pervading the entire industry, and wasting billions of dollars.

The commercial product that will result from a Phase II STTR effort will help real-time software system developers with the following types of software development problems.

1. Although humans will make errors, testing should eliminate errors. Clearly, testing is not always carried out effectively or the *Clementine* would not have failed. There are two separate reasons why testing is not successful. Budget constraints affect testing. This problem is not unique to real-time systems. The existence of a finished product is a very visible result of any development effort. The level of testing that has gone into the product is not as visible, at least not until the product is used and fails. The *Clementine* worked perfectly mapping out the lunar surface and then failed catastrophically. It is impossible for a manager to cut out items which directly lead to the existence of the finished product, but what he can cut is testing.

For real-time systems the testing problem has an extra dimension. Not only must the software work correctly, but in a specified amount of time. At Denver, timing problems resulted in the luggage landing on the tracks instead of inside the cars. The problem with testing real-time systems is especially difficult and vexing and there are cases of real-time systems passing thousands of hours of engineering tests and failing in the field.

2. Project scheduling and management is a problem. An experienced building contractor can inspect a building site and estimate how far the project is from completion. Software is intangible. There is no analog in software development to a visual inspection of the building site. Commercial software rarely arrives on the expected date. Even large and experienced organizations like the FAA and IBM can find themselves hopelessly behind on a project with cost estimates $1 billion too low.

3. Documentation is cumbersome and costly. Creating and managing the design specifications requires discipline. The documentation is a substitute for corporate knowledge of the system that theoretically could be kept in one's head. On long projects the programmers and managers who started the work may not be around by the time the project is over. Looking at our examples, IBM Federal Systems was sold by IBM to Loral, who is now responsible to the FAA for the completion of the AAS project. Compounding the problem was the fact that IBM Federal Systems did not maintain the documentation needed for a project with the scope of the AAS.

4. Programmer skill is also a factor. Anything that could simplify and aid programmers would help eliminate some of the errors and speed the project along.

A.2 The Computer Aided Prototyping System

The Computer Aided Prototyping System (CAPS) has been an ongoing research project at the Naval Postgraduate School (NPS) in Monterey, CA for eight years. The basic thrust of this research project has been to facilitate the design and creation of real-time software with the goal of eliminating errors and saving money. Research funds have come from several sources. Funding has come from the Army Research Office and the Ada Joint Program Office, because of the obvious need for CAPS for a host of military systems. ISTI and Monmouth University have interests in C^3I which led to our involvement with CAPS. Research funds came initially from the National Science Foundation, because CAPS will be of use in the development of non-military systems. Several private sector corporations have provided funds for similar reasons. CAPS does provide a complete real-time system development workstation environment with special emphasis on computer aid to support the aforementioned problem areas of development. It does seem reasonable, that CAPS will be a commercially available product that will be of maximum benefit to software developers. Inside CAPS are the seeds to the solutions of all the problems mentioned in the previous section.

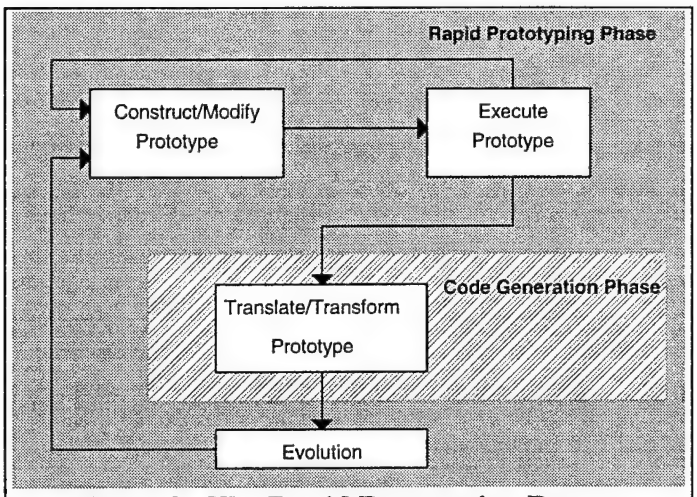


Figure 9. The Rapid Prototyping Process

CAPS is a software engineering tool for developing prototypes of real-time systems. It is useful for requirements analysis, feasibility studies and the design of large embedded systems. CAPS is based on the Prototype System Description Language (PSDL) [Reference 7], which provides facilities for modeling, timing and control constraints within a software system. PSDL is a fifth generation programming language. Project management is provided by CAPS through its development environment, implemented in the form of an integrated collection of tools, linked together by a user-interface.

The CAPS approach offers a true prototyping language and architectural framework that combines the best features of a computer aided software tool and man-in-the-loop simulation. It provides a mechanism for requirements refinement via iterative, build-a-little, test-a-little philosophy, concept confirmation from the rapid creation and test involving the real user, and the creation of code for rapid insertion into real equipment platforms.

In the planned first and second releases of the commercial version, CAPS will provide the following kinds of support to the prototype designer:

1. timing feasibility checking via the scheduler,
2. automated support for software testing including timing via a man-in-the-loop simulation,
3. consistency checking and some automated assistance for project planning, scheduling, designer task assignment, and project completion date estimation via the Evolution Control System,
4. design completion via the editors, and
5. computer-aided software reuse via the software base.

The first element in the list addresses what is perceived as a common problem for the designers of real-time systems. For anything beyond the simplest real-time system, it is difficult to determine if the timing constraints in the design are feasible. The design must contain timing constraints on each module in the code, where each module will perform a simple task, as well as timing constraints on the over-all operation, which may involve the execution of a very large number of modules and an near infinite set of possible circumstances. As part of the PSDL language, the designer will specify the timing constraints as well as other specifications and architectural attributes. The CAPS scheduler attempts to schedule the modules and if the scheduler is unsuccessful, diagnostic messages are given. CAPS does not try every possible scheduling scheme, because that task would take an enormous amount of computing resources. If CAPS fails to schedule the prototype, the designer is not assured that it is impossible to successful

schedule the tasks with the timing constraints as given. If CAPS is unsuccessful at scheduling the prototype, then it is unlikely that the designer will be able to find one on his own. At this point the designer needs either more computer power in his target system, a scaled back design with few tasks or a craftier approach. The mathematical approach of the scheduler will be a breakthrough in real-time system design. We know of no commercial product with this ability, however there are other research groups which have looked into this problem.

Because the timing constraints in the design are in the nature of promises, after the modules are written, it is advisable to see as soon as possible if the promises have been kept. Presently the only way to check the timing is to install the system on the target computer and see if it works. This does not insure success. As mentioned earlier, there have been systems which have undergone extensive and successful testing on the target computer(s), which have failed immediately when put into use, The reason has been ascribed to the fact that there are no methods of conducting timing tests for real-time systems that are guaranteed to include the worst case possibility.

CAPS provides a feature in its execution support system combined with the Hardware Model, which checks the timing of the finished software in a similar fashion as checking it on the target computer. A man-in-the-loop simulation of the prototype is integrated into the CAPS tool, and the ability of each module to finish its task in the specified amount of time is checked using the designer's workstation's CPU, instead of the target computer's CPU and hardware. Those elements of the code that are working too slowly can be identified and corrected in an earlier phase of development.

The man-in-the-loop simulation is an important aspect of the rapid prototyping process. Software is always tested in some sort of simulation, before it is approved. The "man-in-the-loop" simulation that CAPS facilitates is a test of the coding inside the workstation environment with a simulation that includes real-time inputs from the programmer and real-time outputs on the workstation's monitor. For instance, for a C^3I simulation the programmer would be able to see a model of the C^3I displays on his workstation and to start the simulation he could create enemy targets and friendly units using his mouse and keyboard. As the simulation progressed, he could control the targets or produce new ones from his keyboard. CAPS has tools that aid the programmer in the creation of the simulation. By automating the simulation process, CAPS will speed up the testing of the software and allow for better tests.

CAPS generates the coding to tie together the pieces of the system. From the PSDL description of the system, CAPS has a blueprint of the system. If the scheduler is successful in finding a schedule, then the translator creates the coding required to link all of the parts of the system together. The only coding required on the part of the human programmers, is that needed for each individual module, however when the software is transferred to the target computer, the coding created by CAPS to link the modules together may have to be rewritten for several reasons. This rewrite may needed to be optimized to speed execution or to account for special features in the target hardware, that do not exist on the programmer's workstation. The links to the man-in-the-loop simulation must be removed and replaced by links to the target computer's I/O.

Having a copy of the CAPS generated coding as a guide will simplify the task of this final increment of software creation.

Another part of the CAPS system which aides in creating the simulation is an interface editor. The interface editor generates the coding for the man-in-the-loop simulation input and output GUI. This coding will not be part of the target system coding. It interfaces the prototype with the workstation. The programmers will be able to spend their time creating code that will be used in the final system and not creating code for the tests.

The Evolution Control System (ECS) mentioned as the third item in the list of features will manage the design documentation and enable managers to check on the progress of the project on a daily basis. CAPS has an internal copy of the work plan and alerts the manager on the day that some part of the work is late. The CAPS code maintains an internal model of the abilities of each member of the design team and helps the manager to reschedule the project, as the programmers fall behind (or progress faster than) the original schedule. For CAPS to assign each programmer with a skill level, the manager must assign a difficulty factor to each module to be coded. The programmers are rated by CAPS by the speed with which they complete their coding assignments.

The software design documentation is kept in a repository, with both backwards and forwards traceability, i.e. CAPS knows which documents refers to which software modules. When a programmer finishes an assignment, CAPS will automatically start him on the next task and provide on his disk a copy of all the relevant design documents for the next section of code that the programmer is to develop.

CAPS provides configuration management and is integrated with the text editor used to create the software. The configuration management controls both the software in the software database and the documentation in the design database. Old versions of the software and the documentation are frozen.

Also, CAPS maintains database access of reusable code segments from the software database. The software database will contain all of the code written to date for the project. When the coding for a module has to be written, CAPS will first search the database for all of the existing code, which seems to match the requirements of the this module. If the code has already been written, then the programmer does not have to rewrite it. As it is more likely that the previous coding will not provide an exact match, the programmer may still find some code, which is close to his needs. Then the programmer has a much better starting point, than starting from scratch.

We have discovered that students working with CAPS who did not have any knowledge of Ada were able to use the existing Ada modules to create different Ada modules. They took modules which were close in functionality to the task that they needed and made the changes that seemed natural to create new functions. Our conclusion is that to some degree, the rewrite system will allow programmers with lesser skills to create software at a level above their normal abilities.

The purpose of the Evolution Control System and the software database is to provide a continuity of the corporate knowledge of the design project, even as the programmers and managers leave the project taking their knowledge with them.

A.3 Conclusions

The conclusions we hope to illuminate with this appendix are:

1. The history of software development is one of wasted money and effort. We can easily site three recent and dramatic examples of real-time system development failures. These failures illustrate several areas of weakness in the present state-of-the-art of software development.

2. The CAPS research effort has been underway for eight years under the guidance of two distinguished researchers with a internationally recognized reputation.

3. The commercial version of CAPS will address the problems of real-time system software development. Some of the CAPS tools, such as the CAPS scheduler and the Evolution Control System (ECS) are software technology breakthroughs.

REFERENCES

1. Luqi, "Computer-Aided Prototyping for a Command-and-Control System Using CAPS", *IEEE Software*, January, 1992.
2. *Newsweek*, pages 38-39, August 22, 1994.
3. Rifkin, Glenn, *Forbes Supplement*, pages 110-114, August 29, 1994.
4. Nordwall, Bruce D., *Aviation Week & Space Technology*, page 30, March 8, 1993.
5. Nordwall, Bruce D., *Aviation Week & Space Technology*, pages 29-32, October 10, 1994.
6. Gibbs, W. Wayt, "Software's Chronic Crisis", Scientific American, September 1994.
7. Luqi, Berzins V. and Yeh, R., "A Prototyping Language for Real-Time Software," IEEE *Transactions on Software Engineering*, October 1988.